Parallel Pseudospectral Electronic Structure: I. Hartree–Fock Calculations

DAVID CHASMAN, MICHAEL D. BEACHY, LIMIN WANG, RICHARD A. FRIESNER

Department of Chemistry, and Center for Biomolecular Simulation, Columbia University, New York, New York 10027

Received 23 December 1997; accepted 10 February 1998

ABSTRACT: We present an outline of the parallel implementation of our pseudospectral electronic structure program, Jaguar, including the algorithm and timings for the Hartree–Fock and analytic gradient portions of the program. We also present the parallel algorithm and timings for our Lanczos eigenvector refinement code and demonstrate that its performance is superior to the ScaLAPACK diagonalization routines. The overall efficiency of our code increases as the size of the calculation is increased, demonstrating actual as well as theoretical scalability. For our largest test system, alanine pentapeptide [818 basis functions in the cc-pVTZ(-f) basis set], our Fock matrix assembly procedure has an efficiency of nearly 90% on a 16-processor SP2 partition. The SCF portion for this case (including eigenvector refinement) has an overall efficiency of 87% on a partition of 8 processors and 74% on a partition of 16 processors. Finally, our parallel gradient calculations have a parallel efficiency of 84% on 8 processors for porphine (430 basis functions). J Comput Chem 19: 1017–1029, 1998

Keywords: pseudospectral; parallel; Hartree–Fock; gradient; scalable

Correspondence to: R. A. Friesner

Contract/grant sponsor: National Science Foundation; contract/grant number: CHE9217368

Contract/grant sponsor: National Institutes of Health, Division of Research Resources; contract/grant number: P41RR 06892

Contract/grant sponsor: MetaCenter

This article contains Supplementary Material available from the authors upon request or via the Internet at ftp.wiley.com/public/journals/jcc/suppmat/19/1017 or http://journals.wiley.com/jcc/

CCC 0192-8651 / 98 / 091017-13

Introduction

Parallelization of *ab initio* electronic structure methods is an essential task if one wishes to treat large molecules in a cost-effective and timely fashion. The latest generation of parallel machines, such as the IBM-SP2, combine the low cost per processor of workstation technologies with the real-time throughput potential of a large supercomputer, when provided with efficient parallel algorithms. The system software, mathematical libraries, and communications hardware and software on these machines are now such that the parallel implementation of large, complex codes can be accomplished with a reasonable level of human effort. Additionally, the large local disk space of the SP2 render it particularly suitable for quantum chemical computations, where files must invariably be read from disk for each iteration in problems of significant size.

In this study, we describe a parallel algorithm for Hartree–Fock (HF) calculations using pseudospectral (PS) numerical techniques as implemented in the Jaguar suite of *ab initio* electronic structure code.[1] Timing results for both single-point and analytical gradient calculations are presented on the IBM SP2. Our code is written using a standard MPI protocol, however, and is therefore readily portable to other parallel machines. While a number of parallel implementations of HF codes have appeared elsewhere,[2–9] the algorithms employed in the PS methodology are significantly different from those in conventional electronic structure approaches, and hence require a different parallelization strategy. We have shown in a previous publication[10] that PS HF calculations are substantially more efficient than alternative codes such as Gaussian-92 (by factors of three to seven depending upon the basis set and hardware platform) for single-point energies of large systems. Gains are particularly large for high-quality basis sets such as the Dunning cc-pVTZ(-f) basis. Therefore, an effective parallel implementation of the PS HF algorithms in Jaguar will result in a powerful and practical tool that will extend the range of chemical systems that can be profitably studied by *ab initio* electronic structure methods.

A major objective in parallelizing computational chemistry codes is the achievement of scalability; that is, the ability to utilize an arbitrarily large number of processors to solve a given problem with a proportional reduction in wall-clock time. However, for Hartree–Fock calculations, there are fundamental limits on scalability that arise from irreducible aspects of the calculations, such as matrix multiplies and matrix diagonalizations, which are inherently quite difficult to parallelize efficiently. (That is true within the current technology. Methods have been proposed[11] that would do away with the need for diagonalization, although it has not been demonstrated that these methods are better in practice as well as theory.) In addition to matrix multiplies and diagonalizations, there are practical challenges associated with achieving precise load balancing in assembly of the Coulomb and exchange operators. If one's Fock matrix assembly time is sufficiently large compared with these terms, then scalability can be demonstrated for a large number of processors, but this simply means that the original single-node code was highly inefficient. In contrast, our exceptionally efficient single-node performance means that Amdahl's law asserts itself much more quickly in the form of the terms previously mentioned. We have attacked a part of this problem via a parallel implementation of the Lanczos diagonalization algorithm.[12] This allows us to achieve an improvement of three to four times in this segment of the calculation as compared with the ordinarily employed alternative of scalable LAPACK.[13] However, this is insufficient to eliminate some degradation in performance at the four- to 8-node level for the problems we examine here.

Despite this difficulty, we do not believe that this observation has serious practical consequences. In our experience, the vast majority of computationally oriented chemical laboratories are at any one time engaged in the study of large number of molecules that can be trivially distributed across a parallel platform simply by running multiple jobs. For this reason, it is not important, in a practical sense, to attain a theoretically perfect scalability for arbitrarily small problems. Rather, what is crucial about parallelization is the ability to tackle large molecules with an acceptable turnaround time, while not sacrificing efficiency. This leads us to define ''practical scalability'' in a different manner: We consider our approach to possess practical scalability if, for a given problem size, the number of nodes that can be used efficiently (say, with 85–90% of the theoretical maximum performance) is such that the wall-clock time for the job is reduced to an acceptable level (e.g., a few hours for a single-point calculation, a few days for a geometry optimization). Our current imple-

mentation achieves this objective, although quantitative improvements are still possible.

We have organized this study as follows: We first present a brief overview of the pseudospectral Hartree–Fock methodology to facilitate the discussion of its parallelization. We then describe in detail the algorithms we have developed for parallelizing different parts of the code. Results are then presented using two basis sets [6-31G** and Dunning cc-pVTZ(-f)] for several single-point calculations, displaying the scaling of wall-clock time with the number of processors. Results are also presented for gradient calculations using the 6-31G** basis. Finally, we summarize the results and suggest future directions.

Pseudospectral Overview

The pseudospectral assembly of Coulomb and exchange matrix elements has previously been described in several studies,[14–20] and we shall simply present the formulas here without elaboration. The Coulomb matrix element between basis functions $|i\rangle$ and $|j\rangle$ is:

$$J_{ij} = \sum_g Q_i(g) J(g) R_j(g) \tag{1}$$

where the physical-space Coulomb operator, $J(g)$, is given by:

$$J(g) = \sum_g A_{kl}(g) \rho_{kl} \tag{2}$$

Here, $A_{kl}(g)$ is a three-center, one-electron integral (potential integral) representing the field at g due to the product charge distribution of basis functions R_k and R_l at the grid point g and is given by:

$$A_{kl}(g) = \int d^3g' \frac{R_k(g') R_l(g')}{|g - g'|} \tag{3}$$

In the formulas just expressed, $R_j(g)$ is the atomic basis function j evaluated at a grid point g, ρ_{kl} is the usual density matrix, and $Q_i(g)$ is a least squares fitting operator:

$$Q_{ig} = \sum_j \left[S_{ij} [\mathbf{R}^\dagger \mathbf{w} \mathbf{R}]^{-1}_{ji} R^\dagger_{ig} w_g \right] \tag{4}$$

which is designed to fit any right-hand side $R_j(g) A_{kl}(g)$ in the region of space in which atomic basis function $|i\rangle$ has significant density. Here, the matrix $\mathbf{w}$ is the diagonal matrix of grid weights. It has been shown previously[10] that the use of a specialized least squares operator with an optimized fitting basis is considerably more accurate for a given number of grid points than any standardized quadrature scheme. Our most accurate grid for the 6-31G** basis uses 300–400 grid points per atom, whereas our most inexpensive grid uses only 100 grid points per atom.

The exchange matrix elements K_{ij} are given by:

$$K_{ij} = \sum_g Q_i(g) K_j(g) \tag{5}$$

where:

$$K_j(g) = \sum_n A_{jn}(g) \sigma_n(g) \tag{6}$$

is the pseudospectral physical space exchange field. Here, the intermediate quantity $\sigma_n(g)$ is defined as:

$$\sigma_n(g) = \sum_m \rho_{nm} R_m(g) \tag{7}$$

When $|i\rangle$ is a long-range function, the least squares fit has to be carried out over a large portion of the molecule. This leads to a significant loss of accuracy for a fixed number of grid points as well as an increased effort in solving the normal equations. This problem has been addressed with a length scales algorithm, which is described in detail in ref. 10. Briefly, the idea is to avoid matrix element calculations in which a diffuse function must be used in the least squares operator. When at least one function is nondiffuse, this can be trivially accomplished by permuting the i and j indices so the diffuse function is on the right. If both $|i\rangle$ and $|j\rangle$ are diffuse functions, more work is required, but it is possible to express both the Coulomb and exchange matrix elements so that a diffuse least squares operator is used only if all three functions following it are also diffuse.

Parallelization Scheme

There are five major parts of the pseudospectral Hartree–Fock algorithm that require parallelization. These are: (a) the generation of the least squares operators; (b) calculations of the one-, two-, and three-center, two-electron integrals via our GHGP code[21,22]; (c) calculation of the three-center, one-electron integrals; (d) the pseudospectral assembly [eqs. (5) and (1)]; and (e) eigenvector re-

finement. We will present our parallelization scheme in the order in which the computational tasks are carried out in the program.

LEAST SQUARES ASSEMBLY

Before the SCF iterations begin, our least squares code assembles the least squares operator **Q** of eq. (4), which transforms from physical space to spectral space. In practice, not all elements of the **Q** matrix are actually computed, because most basis functions drop off sharply as a function of the distance from their centers. Such functions are classified as short-range functions and are grouped together by atom. The remaining functions are classified as long-range functions, which are placed in a single group. The least squares operator is evaluated individually for each of the $N_{atom} + 1$ groups. The N_{atom} short-range fitting matrices are evaluated for the grid points that are within an adjustable cutoff distance of their atomic centers. The long-range fitting matrix is evaluated across all of the grid points. This decomposition of **Q** allows us to only consider small regions of space for the short-range basis functions. Further, as the system size is increased, it will be possible to employ cutoff distances for even the long-range functions. This implies that the evaluation of the least-squares operators, Q_i, scales as $N \log(N)$ with system size, rather than N^2. The length scales algorithm also improves the accuracy obtained with a given number of grid points.[20]

The set of grid points associated with each atom is broken down into spatially contiguous groups (termed grid blocks), and the evaluation of the matrix elements from the short-range functions then takes place over nested groups of grid types, atoms, grid blocks, and, finally, grid points. First, the matrix elements:

$$[R^{\dagger} wR]_{ij} = \sum_{g} R_i(g) w_g R_j(g) \tag{8}$$

are evaluated for a particular atom and grid type. Here, the indices i and j indicate fitting functions centered on the atom for which the matrix is being evaluated and the g is the grid point index. The cost of this matrix multiplication is $N_{fit}^2 N_{grid}$, where N_{grid} is the total number of grid points and N_{fit} is the number of fitting functions associated with the atom for which $[\mathbf{R}^{\dagger}\mathbf{wR}]$ is being evaluated. (The number of fitting functions for a given atom is generally around two to four times the number of basis functions on that atom, depending on the specific basis set and accuracy of the grid being used.) Once the $[\mathbf{R}^{\dagger}\mathbf{wR}]$ matrix for a given atom and grid resolution has been assembled, each of the corresponding matrices $\mathbf{S}[\mathbf{R}^{\dagger}\mathbf{wR}]^{-1}$ [see eq. (4)] are formed using **S** and $[\mathbf{R}^{\dagger}\mathbf{wR}]$ and by solving for X in:

$$[R^{\dagger} wR] X = S \tag{9}$$

Solving this equation by Cholesky decomposition requires on the order of N_{fit}^3 operations. Each Cholesky decomposition is carried out on the node which computed that particular $[\mathbf{R}^{\dagger}\mathbf{wR}]$ matrix.

The formation of the matrices $\mathbf{S}[\mathbf{R}^{\dagger}\mathbf{wR}]$ for each grid resolution is distributed in the $N_{atom} + 1$ tasks comprised of both the matrix multiplication [eq. (8)] and the Cholesky decomposition [eq. (9)]. After all $\mathbf{S}[\mathbf{R}^{\dagger}\mathbf{wR}]^{-1}$ matrices are completed, the results are broadcast to all the nodes. The ratio of computation to communication is:

$$\frac{N_{fit}^2 N_{grid}}{N_{fit}^2 \log(N_{proc})} \tag{10}$$

Because this ratio is proportional to the number of points in the molecular grid over the log of the number of processors, we can rest assured that this parallelization scheme will be successful.

Next, the final Q matrix is assembled (i.e., the matrix $\mathbf{S}[\mathbf{R}^{\dagger}\mathbf{wR}]^{-1}$ is multiplied by the matrix $\mathbf{R}^{\dagger}\mathbf{w}$). These tasks can be distributed by grid blocks. The computational work in this final matrix multiplication is proportional to $N_{fit}^2 N_{grid}$ and the information that needs to be returned for each **Q** is $N_{fit} N_{grid}$, so the ratio of computation to communication is proportional to N_{fit}, or the number of fitting functions associated with each atom or the long-range group. Because this ratio is roughly constant, and the number N_{fit} may be fairly small, there is no guarantee that the distribution of this final matrix multiplication will be practical. There are, however, a number of options available for this matrix multiplication, depending on the load balance method being used in the pseudospectral Fock matrix assembly. If a static load balance is being used, each processor simply builds **Q** for the grid blocks it has been assigned. For this method, there is no communication cost. For a partially dynamic assignment of grid blocks, each processor builds **Q** for its statically assigned grid blocks and all dynamically assigned grid blocks. In this case, the communication penalty is as noted earlier. One significant gain in either wholly static or partially static assignment of grid blocks is in the distribu-

tion of storage of the final **Q** operators, which total size $N_{basis}N_{grid}$ and are the largest single stored quantity in our PS HF implementation. It is also possible to run a calculation with completely dynamic assignment of grid blocks, for which the full **Q** operators are stored on each node. However, all calculations presented here have used partially dynamic grid block assignment.

MATRIX ASSEMBLY

After the least squares program has generated the matrices **Q**, any standard SCF procedure may be employed to obtain an eigenfunction of the molecular Hamiltonian. The only modification of our method is that the pseudospectral method is used in the construction of the relevant operators. First, the analytical corrections are computed (see ''Analytical Corrections'' subsection). The calculation of the analytical corrections is decomposed using columnar ''strips'' of the density matrix and may be distributed across the processors either statically or dynamically. Next, the Coulomb and exchange matrices are assembled using eqs. (1) and (5). This work is broken into grid block tasks, which correspond to groupings of the index g in eqs. (1) and (5). This grid block work is either distributed statically, dynamically (using the pool of tasks paradigm), or partially dynamically. The partially dynamic or completely static methods win out as the size of the molecule grows for two reasons. First, storage of the **Q** operators is not duplicated. Second, the **Q** operators need not be broadcast to all nodes. For small- to medium-sized molecules, such as porphine, these considerations are not important and the calculation can proceed with dynamic assignment of all grid blocks and analytic strips. We recognize that dynamic assignment of grid blocks through a shared counter is not a truly scalable solution to the load balance problem, but it suffices for our stated goal of reducing cycle times to reasonable levels for large problems.

Because there is no interdependency of the analytical corrections and the pseudospectral assembly work, the two tasks can be overlapped in the dynamic grid block assignment methods. That is, as soon as all of the ''strips'' of analytical corrections have been completed, the idle processors immediately begin the assembly by grid blocks as is indicated in Figure 1. To improve load balance during operator assembly, we have varied the size of the grid blocks so that a variable fraction of the grid blocks is no more than half the size of the

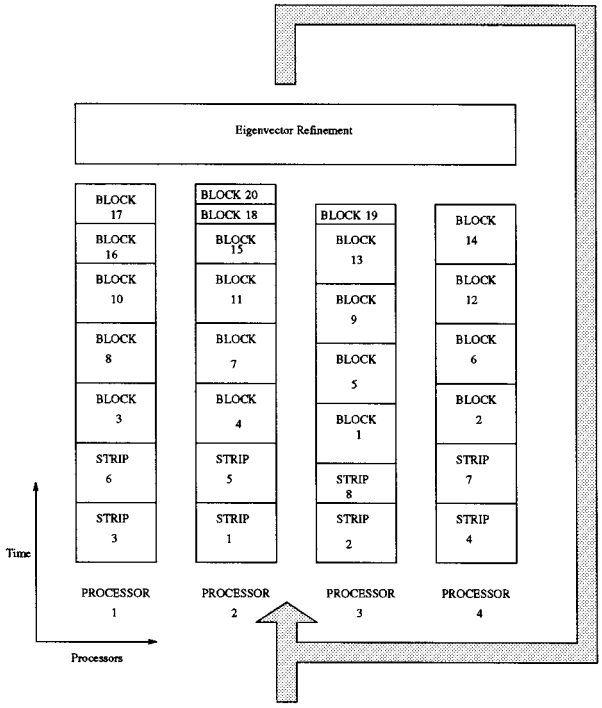

FIGURE 1. Task distribution for a partition of 4 processors, carrying out an SCF calculation on a molecule which requires 8 strips of analytical corrections and 20 grid blocks.

others and we have sorted the grid blocks in descending order of size. This is similar to the reordering of tasks by size done by Lüthi et al.[23] to improve load balance in their SCF parallelization. These two minor modifications are analogous to building up a level wall with stones of varying sizes. By laying down the largest stones first, and then laying down smaller stones, it is more probable that a level wall will be obtained. This point is illustrated in Figure 1.

The final Fock, exchange, and Coulomb matrices are assembled by simply carrying out a global sum of the local copies of the matrices. The local copies of the matrices are identical to the full pseudospectral matrices except that the subscript g in eqs. (1) and (5) has been restricted to the grid blocks that have been allocated to a particular processor. The local replication of these N_{basis}^2 matrices is the only limitation imposed by storage (as opposed to CPU) considerations in our code. (Because all of the data associated with grid points is distributed over processors, it does not constitute a barrier to the size of the system that can be treated.) However, the code is written so that these arrays can be stored on disk and read into the program in

a blocked fashion, so that the penalty in wall-clock time is minimal. Thus, the number of basis functions that can be treated is determined by what can fit on a local disk. The SP2 that we have at Columbia currently has a 2.5-GB local disk, and more current machines are typically equipped with 9-GB or even 16-GB disks. A symmetric 5000×5000 matrix requires about 95 MB of storage, so all of the N_{basis}^2 matrices should easily fit into a gigabyte of storage. With a 9-GB disk drive, a 12,000-basis-function calculation becomes quite feasible. This calculation is significantly larger than any that we are contemplating in the near future (surely such a system is better treated via methods such as mixed quantum mechanics/molecular mechanics) and hence this aspect of the implementation is not as significant in a practical sense. And, although the algorithm presented here uses replicated data structures for practical reasons (ease of implementation), there is no reason that a code with distributed N_{basis}^2 matrices could not be developed.

Once the assembly of the Fock matrix is complete, the eigenvectors are refined using either a serial matrix diagonalization, the ScaLAPACK[13] parallel diagonalization (specifically, subroutine pdsyev, available from Netlib), or the parallel Lanczos-based procedure (see ''Eigenvector Refinement'' subsection). The SCF procedure is then iterated to convergence. All N_{basis}^2 by N_{basis}^2 matrix multiplies are completed with the ScaLAPACK pdgemm subroutine.

Analytical Corrections

As described in the ''Pseudospectral Overview'' section, pJaguar (parallel Jaguar) calculates analytic corrections for the one-center and two-center spectral space terms of the Fock matrix, and for some of the three-center terms as well. Specifically, for the Coulomb matrix elements, we calculate the analytic terms:

$$\sum_{kl} (ij|kl)\,\rho_{kl} \tag{11}$$

for cases in which i, j, k, and l meet certain cutoff criteria and the term $(ij|kl)$ is of the form $(aa|aa)$, $(aa|ab)$, $(aa|bb)$, $(ab|ab)$, or $(aa|bc)$, where a, b, and c indicate the atom upon which the function is centered. Similar correction terms are computed for the exchange operator, as detailed in ref. 10. The corresponding pseudospectral terms in eqs. (1) and (5) for the appropriate i, j, k, and l are subtracted from the pseudospectral elements of J and K.

The calculation of the analytical corrections is decomposed into tasks by breaking the density matrix into columnar strips. In addition, the second (column) index is constrained so that all of the basis functions associated with a given atom are contained in a single strip. The restriction that the basis functions associated with a given atom are not allowed to cross a ''strip boundary'' has the advantage that all terms of the form $(aa|bc)\rho_{bc}$ require only the matrix elements corresponding to atom b, whereas the terms of the form $(aa|bc)\rho_{aa}$ require only the diagonal block ρ_{aa}. This implies that the only density, Coulomb, and exchange matrix entries that are accessed for the calculation of the analytical corrections for a given strip are those contained in the strip itself and those contained in the atom diagonal blocks of the matrix. The subtraction of the pseudospectral correction terms is distributed over the grid blocks, which will be discussed in the next subsection.

The analytical correction terms are evaluated before the Coulomb and exchange matrix elements. Because the analytical corrections are simply added to the final summation of J and K or F, it is unnecessary to sum these terms until the full pseudospectral assembly of the relevant operators has been completed. This allows us to assign the available processors to pseudospectral assembly tasks (described in the next section) as soon as the analytical correction tasks have been completed, without the need for a global synchronization operation. Thus, we are able to overlap the computation of the analytical corrections and the pseudospectral assembly. For the static and partially dynamic grid block assignment methods, the analytic corrections are statically assigned to nodes. The number strips is set to be equal to the number of processors, or some multiple of them. To achieve load balance, the division of strips is done as evenly as possible, within the constraint that functions on a given atomic center remain in the same strip.

The communication costs associated with the analytical corrections are negligible for dynamic grid block assignment, because the only information that needs to be conveyed to the processors is the strip assignment index. For static and partially dynamic grid block assignment, there is no communication cost for the analytical corrections as they are preassigned strips. The cost of communicating the per-processor contributions to the matrices J and K or F may be ignored because the local

copy of these matrices will be incremented and will only be globally summed when the pseudospectral assembly step has been completed.

Pseudospectral Assembly

The first step in each SCF iteration is the evaluation of eq. (7) using the eigenvectors from either the initial guess or the previous iteration. This term is used in the assembly of the exchange operator, as described in eq. (5). Next, the three-center, two-electron terms are calculated for each grid point and for each pair of basis functions k and l, and these terms are used to compute:

$$J(g) = \sum A_{kl}(g)\rho_{kl} \tag{12}$$

and:

$$K_j(g) = \sum_n A_{jn}(g)\sigma_n(g) \tag{13}$$

which are required to evaluate eqs. (1) and (6).

Assembling the contributions from a given block's grid points to the Coulomb matrix element is simply a matter of multiplying $J(g)$ by $R_j(g)$ and then multiplying the result on the left-hand side by $Q_i(g)$. Finally, the contributions must be summed over the grid points for the grid block. Similarly, the grid block's contribution to the exchange matrix element [eq. (5)] is evaluated by multiplying the term just computed, and summing the resulting terms over all grid points in the grid block.

As the Coulomb and exchange contributions from each grid block are evaluated, they are stored locally until all of the grid blocks have been completed. At this point a global sum of the local matrices is done. The cost of collecting any of the final matrices is proportional to $N_{basis}^2 \log(N_{proc})$, and the computational work necessary to assemble the Fock operator is roughly $N_{basis}^2 N_{grid}$. Therefore, the ratio of computation to communication is $N_{grid}/\log(N_{proc})$. This implies that our strategy requires that N_{grid} be significantly larger than the logarithm of the number of processors. This is an extremely modest requirement, as the number of grid points is anywhere from 100 to 400 per atom.

Eigenvector Refinement

The majority of the remaining time in our calculations is taken up by the refinement of the eigenvectors of the Fock matrix, most of which is spent in diagonalization. Because diagonalization with standard QR packages scales as N^3 while pseudospectral Fock matrix assembly scales asymptotically as N^2, pJaguar needs an efficient and well-distributed diagonalization procedure to solve very large electronic structure problems. The serial program Jaguar includes a Krylov-space diagonalization method, which uses the Lanczos algorithm to refine eigenvectors describing occupied orbitals.[12] This technique allows us to use the eigenvectors of the previous SCF iteration as a good approximation to the new eigenvectors. This Krylov-space method allows us to carry out our diagonalizations in a space that is itself a subspace of the occupied subspace. Virtual orbital character is included through the Lanczos algorithm, as described in what follows. The Fock matrix diagonalization for the first SCF iteration is completed with the ScaLAPACK diagonalization routine, as good initial guesses for eigenvectors are needed for the Lanczos algorithm.

In pJaguar, the Lanczos method affords us an important advantage in parallelization: each eigenvector is refined independently of the others, leading naturally to a reasonable parallelization scheme. Although timing results show that overall speedups resulting from parallelization of the eigensolver are small for our test cases, they clearly indicate the feasibility of the approach. As we move to larger problems, we expect that the benefits of the eigensolver parallelization will become apparent.

Our Lanczos diagonalization procedure begins with the construction and diagonalization of the Fock matrix in the space of the N_{occ}-occupied orbitals obtained from either our initial guess or the previous SCF iteration. The transformation of the orbitals to the occupied subspace requires on the order of $N^2 N_{occ}$ operations and the diagonalization in the occupied subspace requires on the order of N_{occ}^3 operations. Because the number of occupied orbitals is typically significantly less than the number of basis functions, N_{basis}, this step, which is carried out on a single node, is considerably less time consuming than the diagonalization of the full Fock matrix. At this point in time, this operation has not been parallelized.

Next, we refine each vector independently using the Lanczos algorithm described in ref. 12, again using the pool of tasks paradigm. Specifically, we construct the Krylov-space representation

in the tridiagonal Lanczos form, using the relations:

$$\begin{aligned}
\mathbf{u}_{j+1} &= \mathbf{G}\mathbf{w}_j \\
\alpha_j &= \mathbf{w}_i^T \mathbf{u}_{j+1} \\
\mathbf{u}'_{j+1} &= \mathbf{u}_{j+1} - \alpha_j \mathbf{w}_j - \beta_j \mathbf{w}_{j-1} \\
\beta_{j+1} &= \left(\mathbf{u}'^T_{j+1}\mathbf{u}'_{j+1}\right)^{1/2} \\
\mathbf{w}_{j+1} &= \mathbf{u}'_{j+1}/\beta_{j+1}, \qquad (14)
\end{aligned}$$

where α_j and β_j are the entries in the tridiagonal matrix, $\mathbf{w_0}$ is the initial guess vector, and $\mathbf{G}$ is the occupied subspace representation of the Fock matrix. As the Krylov space is extended one dimension at a time using eq. (14), the Krylov-space representation of the Fock matrix, which contains the parts of the overall vector space of $\mathbf{F}$ that are most strongly coupled to the original vector (including couplings to the virtual orbitals), is diagonalized and the eigenvector that correlates most strongly with the initial guess vector, $\mathbf{w_0}$, is monitored for convergence. Once we have obtained the new set of approximate eigenvectors, they are orthogonalized using the Gram–Schmidt procedure and the Fock matrix is rediagonalized in the space of the refined eigenvectors.[12] This procedure is repeated until the eigenvectors are converged. Because the individual Krylov-space refinements [eq. (14)] of the eigenvectors require only that the original eigenvector be sent to each slave and then returned, and the computational work required for each of the refinements is dominated by the matrix multiplications on the first line of eq. (14), we see that the ratio of computation to communication will be proportional to $N_{occ}^2/N_{occ} = N_{occ}$, the number of occupied orbitals. Thus, as the number of occupied orbitals increases with system size, the distribution of the Krylov-space-based eigenvector refinement will become more efficient. Currently, the reorthogonalization/rediagonalization is carried out on a single node.

GRADIENT CALCULATIONS

The pseudospectral method for Hartree–Fock gradient calculations is described elsewhere.[24] Here, we briefly sketch the procedure previously reported. The derivative of the Hartree–Fock energy with respect to the nuclear coordinate x_A is:

$$\frac{\delta E_{HF}}{\delta x_A} = \frac{\delta V_{nuc}}{\delta x_A} + 2\sum_{ij}\rho_{ij}\frac{\delta H^0_{ij}}{\delta x_A} - 2\sum_{ij} W_{ij}\frac{\delta S_{ij}}{\delta x_A} + \sum_{ijkl}\rho_{ij}D_{kl}\frac{\delta[2(ij|kl) - (ik|jl)]}{\delta x_A} \qquad (15)$$

where S is the overlapping matrix, W is the energy weighted density matrix $W_{ij} = \sum_\alpha \epsilon_\alpha c_{i\alpha} c_{j\alpha}$, and ϵ_α corresponds to the orbital energy of orbital α. The first three terms are simple to evaluate. The final term:

$$\sum_{ijkl}\rho_{ij}\rho_{kl}\frac{\delta[2(ij|kl) - (ik|jl)]}{\delta x_A} \qquad (16)$$

involves the derivatives of two-electron integrals. These can be expanded as:

$$\frac{\delta(ij|kl)}{\delta x} = (i^x j|kl) + (ij^x|kl) + (ij|k^x l) + (ij|kl^x) \qquad (17)$$

The various terms of eq. (16) may be rewritten as:

$$\begin{aligned}
2\sum_{lk}\rho_{lk}\sum_g Q^x_l(g)R_k(g)\sum_{ij}\rho_{ij}A_{ij}(g) &= 2\sum_{lk}\rho_{lk}\sum_g Q^x_l(g)R_k(g)J(g) \\
&= 2\sum_{lk}\rho_{lk}J^x_{lk} \qquad (18)
\end{aligned}$$

Here, we have used the definitions of $J(g)$ and J_{ij}, which appear in eqs. (1) and (2), respectively. We note that the essential difference between the J^x_{lk} in eq. (18) and J_{ij} in eq. (1) is that $Q_l(g)$ has been replaced by $Q^x_l(g)$, which corresponds to the least squares operator for the derivative function. Although some details have been omitted (e.g., the use of selected analytical integrals to decrease the necessary grid density), it is clear that the decomposition that we applied to our Fock matrix assembly can be applied to our gradient calculations. The calculation of the two-electron integrals has in fact been decomposed into analytical correction ''strips'' and grid block assembly in exactly the same manner that we decomposed our Fock matrix assembly. Fortunately, the computation of the analytical derivatives is constituted almost exclusively by the ''strip'' correction and grid block assembly, which can both be carried out with high parallel efficiency. This is reflected in our timings,

which demonstrate a high efficiency, even for the smallest molecule used for testing our SCF code.

Results and Discussion

In Tables I–V and Figures 2–6, we present results obtained for our code developed using MPL/MPI on the Cornell Theory Center SP2. The timings presented here were for runs conducted on the Columbia Chemistry/Lamont Doherty Geophysical Observatory SP2 on a uniform set of 256-MB thin nodes.

We note first that, in the present implementation, parallelization of the least squares fitting routine is not particularly efficient. However, this constitutes a small fraction of the CPU time, and hence becomes important only for a large number of processors. In contrast, parallelization of Fock matrix assembly is the most efficient aspect of the calculation. Timings for the entire SCF routine are not as scalable as the matrix assembly portion, reflecting the small utility routines that are difficult to parallelize.

We note very good speedup of the matrix assembly as a function of the number of processors.

TABLE I.
Hartree–Fock SCF Timings (Seconds) and Parallel Performance for Porphine in the 6-31G Basis (430 Basis Functions).**

N_{proc}	Least squares	Speedup	Matrix assembly	Speedup	SCF	Speedup	Overall	Speedup
1	150.4	1.00	2138.6	1.00	2220.5	1.00	2431.9	1.00
2	76.4	1.97	1091.7	1.96	1165.8	1.90	1297.6	1.87
4	44.6	3.37	570.5	3.75	640.0	3.47	738.2	3.29
8	36.8	4.09	308.8	6.93	383.1	5.80	476.3	5.11
16	29.8	5.05	199.7	10.71	281.5	7.89	371.9	6.54

TABLE II.
Hartree–Fock Timings (Seconds) and Parallel Performance for BPH in 6-31G Basis (740 Basis Functions).**

N_{proc}	Least squares	Speedup	Matrix assembly	Speedup	SCF	Speedup	Overall	Speedup
1	430.8	1.00	15,541.0	1.00	16,097.9	1.00	16,729.6	1.00
2	220.2	1.96	7829.3	1.98	8265.1	1.95	8687.4	1.93
4	140.1	3.07	4100.6	3.79	4485.4	3.59	4832.0	3.46
8	115.7	3.72	2179.1	7.13	2578.7	6.24	2898.5	5.77
16	105.5	4.08	1243.8	12.49	1651.0	9.75	1962.7	8.52

TABLE III.
Hartree–Fock SCF Timings (Seconds) and Parallel Performance for Alanine Pentapeptide in cc-pVTZ(-f) Basis (818 Basis Functions).

N_{proc}	Least squares	Speedup	Matrix assembly	Speedup	SCF	Speedup	Overall	Speedup
1	1807.7	1.00	28,379.6	1.00	29071.5	1.00	31,147.4	1.00
2	922.0	1.96	14,214.0	2.00	14,824.3	1.96	16,025.2	1.94
4	508.0	3.56	7250.0	3.91	7722.3	3.76	8505.8	3.66
8	463.3	3.90	3764.0	7.54	4184.0	6.95	4925.6	6.32
16	438.6	4.12	2011.1	14.11	2447.3	11.88	3160.9	9.85

TABLE IV.
Hartree–Fock Analytical Gradient Timings (Seconds) and Parallel Performance for Porphine in 6-31G Basis.**

N_{proc}	One-electron gradients (serial)	Two-electron gradients	Gradient speedup	Full geometry iteration	Overall speedup
1	16.9	1845.2	1.00	6137.5	1.00
2	16.8	918.1	1.99	3186.5	1.93
4	16.9	477.1	3.76	1716.4	3.58
8	16.9	260.5	6.71	1037.0	5.92
16	16.8	137.3	11.97	780.0	7.87

TABLE V.
Hartree–Fock Analytical Gradient Timings (Seconds) and Parallel Performance for BPH in cc-pVTZ(-f) Basis.

N_{proc}	One-electron gradients (serial)	Two-electron gradients	Gradient speedup	Full geometry iteration	Overall speedup
1	51.3	6156.6	1.00	26898.5	1.00
2	51.3	3117.9	1.96	13955.8	1.93
4	51.6	1627.5	3.70	7692.7	3.50
8	51.5	872.6	6.72	4670.8	5.76

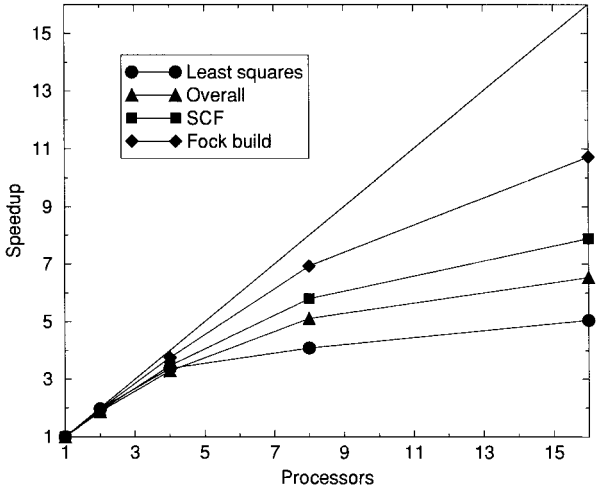

FIGURE 2. Parallel speedups for porphine in the 6-31G** basis.

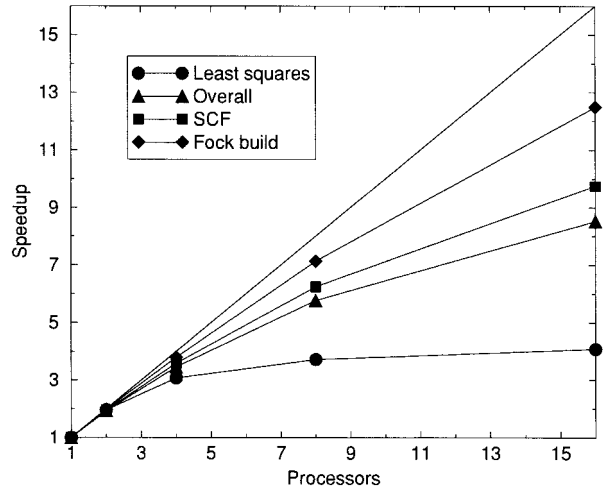

FIGURE 3. Parallel speedups for BHP in the 6-31G** basis.

This is entirely consistent with the performance model put forth earlier. At 94% efficiency, the individual Fock build times for porphine on four processors range from 36 to 145 seconds depending on the grid used and whether or not analytic corrections are employed for that iteration. The average Fock build time for this case is 63 seconds. For BPH on eight processors Fock build times range from 70 to 284 seconds and average 167 seconds, at 89% efficiency. The individual Fock build times for alanine pentapeptide have not been reduced to absolute times, which are as low while retaining 90% efficiency. At 88% efficiency (16 processors), however, absolute times range from 187 to 322 seconds and average 222 seconds. The major contribution to inefficiencies in the Fock matrix assembly is from poor load balancing. For porphine on four processors, perfect load balance would have saved approximately 15.9 seconds of wall-clock time and yielded a Fock matrix assem-

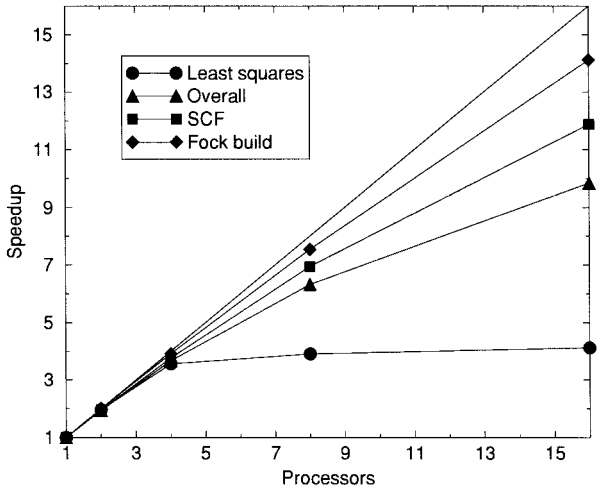

FIGURE 4. Parallel speedups for alanine pentapeptide in the cc-pVTZ(-f) basis.

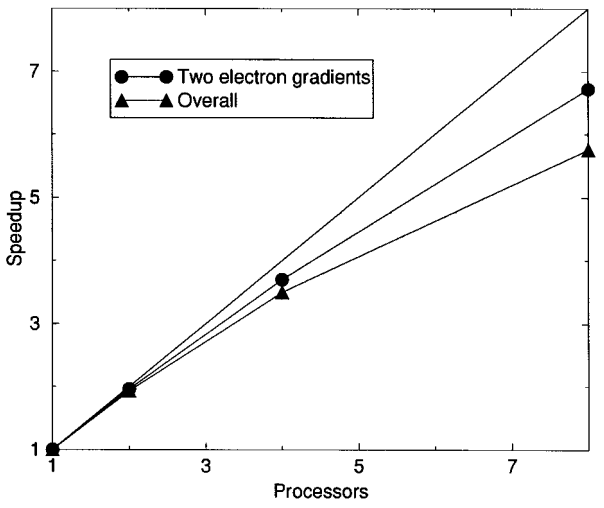

FIGURE 6. Parallel speedups for BHP gradient calculations in the 6-31G** basis.

bly speedup of 3.86. For 8 and 16 processors, perfect load balance would yield speedups of approximately 7.5 and 14.0. The main reason for poor load balance here is an imbalance in the analytic correction times. This could probably be masked by increasing the number of dynamically assigned grid blocks.

After load balancing in the Fock matrix build, we find that the greatest single hindrance to our overall performance results from the parallel eigenvector refinement. The speedups for porphine eigenvector refinement are summarized in Figure 7. A total of 27.4 seconds was spent in diagonalization (9.3 seconds in the first iteration full diagonalization, 18.1 seconds in the remaining seven Lanczos eigenvector refinements) for the single-processor case. For four nodes, total wall-clock time was 17.8 seconds, and total speedup was 1.5. For 8 and 16 nodes, total times of 15.6 and 16.5 seconds were observed, respectively. We can compare the performance for the first iteration diagonalization (done with ScaLAPACK) with the performance in the Lanczos iterations to get an idea of how much of an improvement is achieved by using the parallel Lanczos algorithm. If all iterations used a parallel diagonalization instead of the Lanczos algorithm, we would expect a total time of $8 \times 9.3 = 74.4$ seconds for eigenvector refinement. For 4, 8, and 6 processors, total time would be approximately 74, 62, and 68 seconds, respectively. Thus, we see that we gain a factor three or

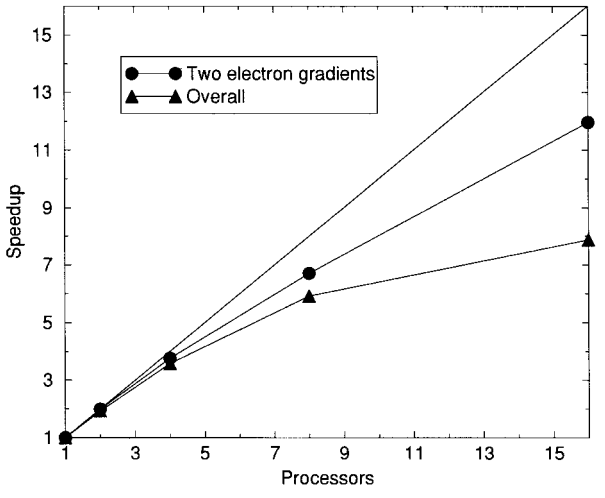

FIGURE 5. Parallel speedups for porphine gradient calculations in the 6-31G** basis.

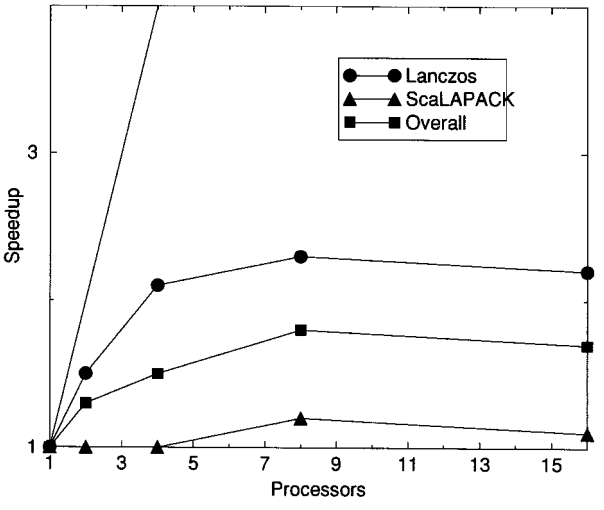

FIGURE 7. Eigenvector refinement speedups for the porphine 6-31G** calculation.

four in wall-clock time with the Lanczos algorithm. Although the speedups for both ScaLAPACK and our parallel Lanczos implementation are rather poor, the Lanczos speedups are higher for each case. More importantly, the serial diagonalization times in the Lanczos method are much lower.

Outside of the Fock matrix build and eigenvector refinement, the remaining parallelizable problem results from N_{basis} squared matrix multiplies. With the parallelization of these matrix multiplies, all major portions of the code that can be effectively parallelized, have been. Some of the portions that remain serial include the DIIS convergence scheme, I/O on replicated data, and calculation of basis function pseudo overlaps.

It can be seen from the data in Tables I–V that we have accomplished our stated goal of achieving low wall-clock time while maintaining high efficiency in comparison to one-node calculations. If we examine only cases where $\sim 90\%$ efficiency is maintained, we see that all Hartree–Fock calculations examined here are completed in less than 2.5 hours. We also see that the number of nodes that can be efficiently employed increases as the problem size increases. For porphine, with 430 basis functions, it is only possible to use two nodes at 94% efficiency, achieving a total wall-clock time of 22 minutes. Increasing the problem size to 740 basis functions in BPH allows 4 nodes to be utilized at 87% with a total wall-clock time of 1 hour and 21 minutes. And, again increasing the problem size to 818 basis functions in alanine pentapeptide allows four nodes to be utilized at 92% efficiency with a total time of 2 hours and 22 minutes.

In our gradient calculations we achieve a higher efficiency than in the corresponding Hartree–Fock calculation. For porphine, four nodes can be utilized efficiently for the gradient calculation where only two nodes can be used in the Hartree–Fock calculation. The total efficiency (including the least squares, SCF, and one-electron portions of the program) at four nodes for a single geometry iteration is 89%. For geometry optimization problems where the result is urgently desired, one might choose to accept lower efficiency in return for faster turnaround. In the case of porphine, running on eight nodes leads to 74% overall efficiency.

Conclusion

In this article, we have presented our parallel algorithm for pseudospectral electronic structure calculations and proven its viability by showing that the ratio of computation to communication is much larger than 1 for each of the tasks that we have successfully parallelized. In addition, we have provided timing data for molecules having 430, 740, and 818 basis functions. Further, we have detailed results for our parallel gradient code and provided timings for a system with 430 basis functions. Despite its deficiencies, our code can currently carry out a gas-phase geometry optimization cycle (i.e., a Hartree–Fock calculation, HF-gradient calculation, and geometry step) on porphine (430 basis functions) in roughly 20 minutes on an eight-processor SP2 partition.

In applications of electronic structure calculations, work is routinely carried out for a diverse mix of chemical problems, each of which requires investigation of a substantial number of molecules. Thus, overall throughput is of comparable importance to turnaround time for an individual job, and one has to strike a balance between these two objectives. As noted earlier, the pseudospectral method is inherently more efficient than corresponding analytical approaches, and it is therefore probable that parallelization of an analytical Hartree–Fock code on the IBM-SP2 would efficiently use more nodes than we have. However, greater parallel scalability at the expense of much greater computational cost is pointless. Our design strategy has been to preserve the high efficiency of our single-node performance, and to carry out parallelization within this constraint.

Parallelization of the Hartree–Fock functionality is only the beginning step in the development of parallel pseudospectral algorithms. It is, however, the most critical one because the algorithms for electron correlation (GVB, MP2, GVB-LMP2, DFT) all utilize the core Hartree–Fock computational functions, such as assembly of Coulomb and exchange operators. Our local MP2 algorithm has already been parallelized, as is reported in part II of this series.[25] Efficiencies that are superior to those reported here are obtained, as many of the small auxiliary routines used for SCF convergence are not relevant to an LMP2 calculation. We expect that similar efficiencies will be obtained for the additional correlation methods just listed.

Supplementary Material

The structure files for each of the molecules used in timing runs in this article are available in XMol's xyz format.

References

1. *Jaguar v3.0*, Schrödinger, Inc., Portland, OR, 1997.
2. M. Feyereisen and R. A. Kendall, *Theor. Chim. Acta*, **84**, 289 (1993).
3. H. P. Lüthi and J. Almlöf, *Theor. Chim. Acta*, **84**, 443 (1993).
4. S. Vogel, J. Hutter, T. H. Fischer, and H. P. Lüthi, *Int. J. Quantum Chem.*, **45**, 665 (1993).
5. S. Brode, H. Horn, M. Enrig, D. Moldrup, J. E. Rice, and R. Ahlrichs, *J. Comput. Chem.*, **14**, 1142 (1993).
6. D. Bernholdt, E. Apra, H. Fruchtl, M. Guest, R. Harrison, R. Kendall, R. Kutteh, X. Long, J. Nicholas, J. Nichols, H. Taylor, A. Wong, G. Fann, R. Littlefield, and J. Nieplocha, *Int. J. Quant. Chem. Quantum Chem. Symp.*, **29**, 475 (1995).
7. M. Guest, P. Sherwood, and J. van Lenthe, *Theor. Chim. Acta*, **84**, 423 (1993).
8. I. T. Foster, J. L. Tilson, A. F. Wagner, R. L. Shepard, R. J. Harrison, R. A. Kendall, and R. J. Littlefield, *J. Comput. Chem.*, **17**, 109 (1996).
9. R. J. Harrison, M. F. Guest, R. A. Kendall, D. E. Bernholdt, A. T. Wong, M. Stave, J. L. Anchell, A. C. Hess, R. J. Littlefield, G. L. Fann, J. Nieplocha, G. S. Thomas, D. Elwood, J. L. TIlson, R. L. Shepard, A. F. Wagner, I. T. Foster, E. Lusk, and R. Stevens, *J. Comput. Chem.*, **17**, 124 (1996).
10. B. H. Greeley, T. V. Russo, D. T. Mainz, R. A. Friesner, J.-M. Langlois, W. A. Goddard III, R. E. Donnelly Jr., and M. N. Ringnalda, *J. Chem. Phys.*, **101**, 4028 (1994).
11. R. Shepard, *Theor. Chim. Acta*, **84**, 343 (1993).
12. W. T. Pollard and R. A. Friesner, *J. Chem. Phys.*, **99**, 6742 (1993).
13. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1997.
14. R. A. Friesner, *Chem. Phys. Lett.*, **116**, 39 (1985).
15. R. A. Friesner, *J. Chem. Phys.*, **85**, 1462 (1986).
16. R. A. Friesner, *J. Chem. Phys.*, **86**, 3522 (1987).
17. R. A. Friesner, *J. Phys. Chem.*, **92**, 3091 (1988).
18. M. N. Ringnalda, Y. Won, and R. A. Friesner, *J. Chem. Phys.*, **92**, 1163 (1990).
19. J. M. Langlois, R. P. Muller, T. R. Coley, W A. Goddard III, M. N. Ringnalda, Y. Won, and R. A. Friesner, *J. Chem. Phys.*, **92**, 7488 (1990).
20. M. N. Ringnalda, M. Belhadj, and R. A. Friesner, *J. Chem. Phys.*, **93**, 3397 (1990).
21. P. Gill, M. Head-Gordon, and J. Pople, *J. Chem. Phys.*, **94**, 5564 (1990).
22. P. Gill, M. Head-Gordon, and J. Pople, *Int. J. Quantum. Chem.*, **S23**, 269 (1989).
23. H. P. Lüthi, J. E. Mertz, M. W. Feyereisen, and J. E. Almlöf, *J. Comput. Chem.*, **13**, 160 (1992).
24. Y. Won, J.-G. Lee, M. N. Ringnalda, and R. A. Friesner, *J. Chem. Phys.*, **94**, 8152 (1991).
25. M. D. Beachy, D. Chasman, R. A. Friesner, and R. B. Murphy, *J. Comput. Chem.*, **19**, 1030 (1998).